УДК 021.8 + 025.1
ББК 78.34

МЕТОД АВТОМАТИЧЕСКОГО ПЛАНИРОВАНИЯ СОВОКУПНОСТИ ТРАЕКТОРИЙ ДЛЯ НАВИГАЦИИ БЕСПИЛОТНЫХ ТРАНСПОРТНЫХ СРЕДСТВ

Яковлев К. С.[1], Баскин Е. С.[2], Андрейчук А. А.[3]
(*Институт системного анализа Федерального исследовательского центра «Информатика и управление» РАН*)

Рассматривается задача планирования совокупности альтернативных траекторий беспилотного транспортного средства (*БТС*)*. Эта задача является ключевой подзадачей более общей проблемы – построения множества неконфликтных траекторий для коалиции БТС. Предлагается новый метод планирования, косвенно учитывающий ограничения на динамику движения транспортного средства, а также ряд его модификаций. Приводятся результаты экспериментальных исследований. В качестве модельной рассматривается задача планирования траектории маловысотного полета беспилотного летательного аппарата мультироторного типа в городских условиях.*

Ключевые слова: беспилотное транспортное средство, беспилотный летательный аппарат, интеллектуальная система управления, навигация, планирование, планирование траектории, альтернативные траектории, построение пути, поиск пути, поиск *k* кратчайших путей, эвристический поиск, геометрические ограничения, LIAN, MultiLIAN.

---

[1] *Константин Сергеевич Яковлев*, *кандидат физико-математических наук* (*yakovlev@isa.ru*).
[2] *Егор Сергеевич Баскин*, *аспирант ИСА РАН* (*baskin@isa.ru*)
[3] *Антон Андреевич Андрейчук*, *студент* (*andreychuk@mail.com*).

1. Введение

В настоящее время наблюдается значительное повышение интереса к управлению комплексами взаимодействующих сложных технических систем. Так, в интеллектуальной робототехнике активно развивается направление, связанное с автоматизацией управления коллективов беспилотных транспортных средств (в терминологии искусственного интеллекта – интеллектуальных агентов), в частности – малых беспилотных летательных аппаратов мультироторного типа, таких как Parrot Ar.Drone [14, 31], Mikrokopter [28], AscTec Hummingbird [11]. При этом, когда идет речь об интеллектуальных системах управления сложными техническими объектами [4, 9], подразумевается разделение управления на так называемые уровни управления [8]. Обычно выделяются три уровня управления – стратегический, тактический и реактивный. Навигационные задачи, в том числе рассматриваемая в статье задача планирования траектории, относятся к тактическому уровню. Модель мира на тактическом уровне содержит все необходимые для построения траектории пространственные данные (и обычно является графовой моделью). Координаты целевой области передаются на тактический уровень управления со стратегического [3].

Традиционно в искусственном интеллекте (ИИ) задача планирования траектории рассматривается как задача поиска пути на графе и разделяется на две подзадачи: построения графа и поиска пути на этом графе. Известно достаточно много различных графовых моделей, извлекаемых из пространственных данных, подходящих для решения задачи планирования – см., например, обзор в [5]. Также в ИИ известно большое число алгоритмов поиска пути на графе (в основном – эвристических) – A* [19], Theta* [29], R*[27], JPS [18] и др. Эти алгоритмы подразумевают поиск единственного пути на графе, следовательно, подходят для автоматизации управления одним агентом. Для построения согласованного множества неконфликтных траекторий используются методы, использующие вышеуказанные алгоритмы в качестве отдельных составных частей. Другой важной компонентой являются алгоритмы, реализующие стратегии разрешения конфликтов, которые мож-

но разбить на два класса: централизованные и децентрализованные. При централизованном подходе используется единая пространственная модель местности и информация обо всех агентах коллектива, конфликты устраняются на этапе построения траекторий. Такой поход является весьма ресурсоемким, так как пространство поиска растет экспоненциально с числом вовлеченных агентов [25, 33], и его использование на практике весьма ограничено. При децентрализованном подходе считается, что отдельные агенты могут обладать различными пространственными моделями, согласование которых возможно лишь при соблюдении определенных условий (например, лишь при нахождении агентов на определенном расстоянии друг от друга, необходимом для установления связи), поэтому отдельные агенты строят траектории достижения своих целей независимо друг от друга, а потенциально возникающие конфликты разрешаются на этапе выполнения. К подобным алгоритмам можно отнести, например, FAR [34], WHCA* [32], MAPP [35]. Важной компонентой этих алгоритмов является построение альтернативных траекторий для отдельных агентов. Именно за счет наличия таких траекторий и устраняются конфликты. То есть сначала для каждого агента строится множество альтернативных траекторий, а затем (на этапе разрешения конфликтов) эти траектории согласуются между собой по определенным правилам, и за каждым агентом закрепляется уже вполне определенная траектория из построенного ранее множества. Таким образом, большую важность и актуальность приобретает задача построения совокупности траекторий для отдельного агента, решению которой и посвящена данная работа.

Дальнейшее изложение организовано следующим образом: в разделе 2 описывается рассматриваемая задача и описывается её формальная постановка (задача поиска множества путей на графе особой структуры), в разделе 3 описываются и анализируются существующие методы решения, предлагается новый метод и его модификации, раздел 4 посвящен экспериментальному исследованию предложенных алгоритмов.

2. Постановка задачи

В качестве модельной задачи будет рассматриваться задача планирования совокупности различных траекторий для малого беспилотного летательного аппарата вертикального взлета и посадки мультироторного типа (например, AscTec Hummingbird), осуществляющего маловысотный полет в городских условиях. Полет осуществляется с фиксированной скоростью, в одной плоскости, ниже уровня высотных строений. Каждое строение представляет собой замкнутый многоугольник, координаты вершин которого известны. В качестве источника пространственных данных используется открытая геоинформационная база данных – OpenStreetMaps (OSM) [30].

В работе [7] представлен поход к планированию траектории БЛА в описываемых условиях, опирающийся на следующие положения. Ограничения на динамику движения объекта управления, изначально заданные в виде системы дифференциальных уравнений (модель динамики БЛА), могут быть сведены к геометрическим ограничениям, накладываемым на форму траектории. Последние являются ограничениями на максимальный угол отклонения сегментов траектории, т.е. «считается, что реализуемая (для заданного режима полета) траектория представляет собой последовательность отрезков таких, что угол отклонения между любыми смежными отрезками последовательности не превышает (по модулю) некоторого фиксированного значения $\boldsymbol{\alpha}$» [7]. Для построения траектории с учетом геометрических ограничений используются графы особой структуры – графы регулярной декомпозиции (ГРД), они же МТ-графы (метрические топологические графы), они же в англоязычной терминологии grids.

Заметим, что указанный подход к учету динамики объекта управления при планировании траектории не является единственным. Так, когда речь идет об объектах, обладающих достаточно простой динамикой движения – например, автомобилях – находят широкое применение методы поиска в пространстве управлений. Граф поиска в этом случае строится итерационно с помощью параметризованного стохастического моделирования, фаза поиска пути на графе совмещается с фазой построения

графа [26]. Для проверки связности вершин графа на каждой итерации алгоритма выполняется моделирование следования по соответствующей траектории. Это весьма ресурсоемкий процесс, временные затраты на осуществление которого напрямую зависят от сложности модели динамики движения объекта управления. Именно поэтому подобный подход обычно применяется для планирования траектории транспортных средств, обладающих достаточно простой моделью движения – в основном для колесных [2, 12], а не для БЛА мультироторного типа.

Существуют также альтернативные способы перехода от ограничений на динамику движения объекта управления к геометрическим ограничениям. Так весьма распространен подход, когда граф, моделирующий окружающую среду, содержит не только информацию о возможных положениях объекта в пространстве, но и о его ориентации (обычно дискретизированной до нескольких десятков значений). При этом вопрос об определении ребёр графа решается эвристически – обычно исследователями определяется набор правил перехода, запрещающих «резко» менять пространственную ориентацию объекта [24]. Поиск пути на подобных графах – гораздо менее трудоемкий процесс, чем упомянутый выше поиск пути в пространстве управлений. Развитием этой идеи и является подход, принятый в данной работе, когда считается, что исполнимая траектория состоит из последовательности прямолинейный секций, таких что угол отклонения между двумя смежными секциями не превышает заданного значения. Впервые алгоритм планирования с учетом ограничения на угол отклонения был представлен в [23] и апробирован в составе системы управления беспилотным катером. К сожалению, предложенный алгоритм не является полным, т.е. не гарантирует отыскание пути (даже если путь существует). Для устранения этого недостатка в [7] был предложен алгоритм LIAN, который является полным для вполне определенного класса задач. Применимость LIAN (и его модификаций) для решения задач планирования траектории маловысотного полета в городских условиях показана в [36].

Итак, задача планирования совокупности траекторий для БЛА мультироторного типа рассматривается в данной работе как задача поиска k путей на графе особой структуры – ГРД/МТ-

графе. При этом (для выдерживания ограничений на динамику движения БЛА) искомые пути должны соответствовать ограничениям на максимальный угол отклонения. Формальная постановка задачи приведена ниже.

Будем считать, что ГРД/МТ-граф – это четверка

Gr = <*A, ADJ, d, los*>, где:

A – множество клеток (вершин), представляющее собой матрицу $A_{m\times n} = \{a_{ij}\}$: $a_{ij} = 0 \vee a_{ij} = 1$, $\forall i, j$: $0 \le i < m$, $0 \le j < n$, $m, n \in \boldsymbol{N}$; клетку будем называть проходимой, если $a_{ij} = 0$; непроходимой, если $a_{ij} = 1$;

$ADJ \subseteq A \times A$ – отношение, задающее смежность на множестве клеток (множество смежных клеток);

d: $A \times A \to \Re$ – метрика на множестве *A* (функция задающая способ определения расстояния между клетками);

los: $A^+ \times A^+ \to \{true, false\}$, где $A^+ = \{a_{ij} \mid a_{ij} \in A,\ a_{ij} = 0\}$ – функция видимости (от англ. line-of-sight).

Здесь и далее будем периодически использовать запись $a(i; j)$, или просто $(i; j)$, для обозначения клетки (вершины) a_{ij}. Индексы *i, j* – координаты клеток (вершин). В тех случаях, когда координаты не важны, будем обозначить клетки буквами латинского алфавита: *a, b, c* и т.д.

Две различные клетки $(i_1; j_1)$ и $(i_2; j_2)$ будем считать смежными, если $|i_1 - i_2| \le 1 \vee |j_1 - j_2| \le 1$. Фактически смежным клеткам соответствуют смежные элементы матрицы $A_{m\times n}$.

Будем считать, что функция *d* определена следующим образом:

$$(1)\quad d(a_{ij}, a_{lk}) = \sqrt{(l-i)^2 + (k-j)^2}\ ,$$

т.е. для измерения расстояний используется евклидова метрика.

Нуль-траекторией *ntr*(*a, b*) между двумя проходимым клетками *a, b* будем называть последовательность клеток, построенную по алгоритму Брезенхема [13]. Нуль-траектории соответствует отрезок прямой на плоскости, соединяющий центры соответствующих клеток – см. рис. 1.

Будем также считать, что функция видимости – *los* – возвращает *true* только в том случае, если соответствующая нуль-траектория не содержит непроходимых клеток.

Упорядоченную пару клеток будем называть секцией и обозначать $e = \langle a, b\rangle$. Длиной секции будем называть величину: $len(e) = len\langle a, b\rangle = d(a, b)$. Секция $\langle a, b\rangle$ проходима, если $los(a, b) = true$.

Две секции, имеющие хотя бы одну общую клетку $e_1 = \langle a_{ij}, a_{lk}\rangle$, $e_2 = \langle a_{lk}, a_{vw}\rangle$, будем называть смежными. Углом отклонения секции e_2 от e_1 – $\alpha(e_1, e_2)$ – будем называть модуль угла между векторами $\overrightarrow{a_{ij}a_{lk}}$ и $\overrightarrow{a_{lk}a_{vw}}$, координаты которых равны $(l - i, k - j)$ и $(v - l, w - k)$ соответственно (см. рис. 1).

Пусть зафиксированы две клетки графа: начальная – s, и целевая – g. Путем из s в g будем считать последовательность смежных, проходимых секций: $\pi(s, g) = \{e_1, \ldots, e_v\}$: $e_1 = \langle s, x\rangle$, $e_v = \langle z, g\rangle$. Аналогично можно определить путь как последовательность клеток. Длина пути – сумма длин соответствующих секций: $len(\pi) = len(e_1) + \ldots + len(e_v)$. Величину $\alpha_m(\pi) = max\{\alpha(e_1, e_2), \alpha(e_2, e_3), \ldots, \alpha(e_{v-1}, e_{v-1})\}$ будем называть максимальным углом отклонения пути π.

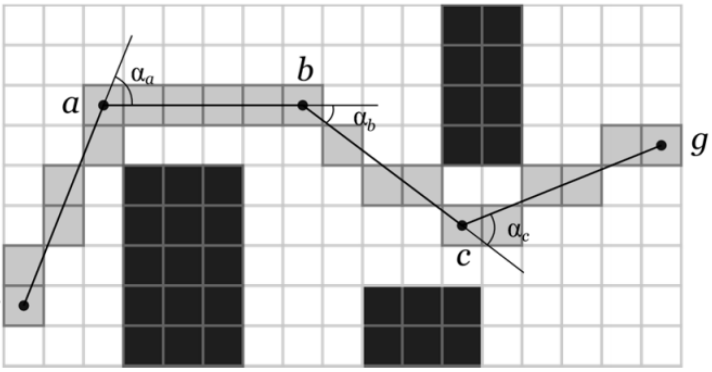

Рис. 1. Задача планирования траектории как задача поиска пути на графе особой структуры. $\pi = \{s, a, b, c, g\}$ – путь, состоящий из проходимых секций (обозначены линиями): $\langle s, a\rangle$, $\langle a, b\rangle$, $\langle b, c\rangle$, $\langle c, g\rangle$. Углы отклонения секций обозначены как α_a, α_b, α_c. Серым цветом обозначены клетки, найденные по алгоритму Брезенхема. Темным цветом обозначены непроходимые клетки

Два пути $\pi_1(s, g)$ и $\pi_2(s, g)$ будем считать различными, если они различаются хотя бы парой секций (или, что то же самое, хотя бы одной вершиной).

Задача планирования совокупности траекторий формально представляет из себя пятерку

PTask = <**Gr**, *s*, *g*, $\boldsymbol{\alpha_m}$, ***k***>

и формулируется следующим образом. Задан ГРД/МТ-граф **Gr** и зафиксированы две вершины: начальная – *s* и целевая – *g*. Необходимо найти ***k*** путей $\pi_1, \ldots, \pi_k$, таких что $\alpha_m(\pi_i) \le \boldsymbol{\alpha_m}$ $\forall \pi_i$: $i = 1, ..., k$.

3. Методы решения

3.1. ОБЗОР МЕТОДОВ ПОИСКА К ПУТЕЙ НА ГРАФАХ

Задача планирования совокупности различных траекторий, сформулированная выше как задача поиска нескольких путей на графе специального вида, в современной литературе чаще всего рассматривается как задача поиска *k* кратчайших путей. При этом рассматриваются пути, которые различаются по крайней мере одной вершиной.

Одним из наиболее известных алгоритмов, решающих задачу поиска *k* кратчайших путей, является алгоритм EA (Epstein's algorithm) описанный в работе [16]. Алгоритм начинают работу с того, что строит кратчайший путь из начальной вершины до всех остальных вершин в графе с помощью алгоритма Дейкстры [15]. Временная оценка этой операции $O(m + n \log n)$, где n – количество вершин, m – количество ребер в графе. На втором шаге алгоритм EA строит граф путей $P(G)$. Граф $P(G)$ в неявном виде содержит в себе искомые k кратчайших путей и для их построения необходимо еще раз применить алгоритм Дейкстры, что потребует ещё $O(k \log k)$ времени. В работе [17] был описан более эффективный способ извлечения k кратчайших путей из $P(G)$, имеющий временную сложность $O(k)$. Таким образом, итоговая оценка асимптотической сложности алгоритма EA составляет $O(m + n \log n + k)$. Данная оценка является самой эффективной для алгоритмов поиска k кратчайших путей.

В работе [22] представлен алгоритм REA (Recursive Enumeration Algorithm). На первом шаге, как и алгоритм EA, REA

строит кратчайший путь из начальной вершины до каждой вершины в графе. Однако на втором шаге алгоритм не строит граф $P(G)$. Вместо этого алгоритм REA работает рекурсивно и для поиска каждого последующего пути использует предыдущий путь. Это позволяет ему быстрее отыскивать решения во многих практических задачах, несмотря на то, что асимптотическая оценка сложности REA выше, чем у EA.

В работе [21] описан алгоритм LVEA (lazy version of Epstein's algorithm). Данный алгоритм имеет ту же асимптотическую оценку сложности, что и алгоритм EA, однако на практике работает быстрее. Значительную часть времени алгоритм EA тратит на то, чтобы построить граф $P(G)$. Алгоритм LVEA строит лишь часть этого графа, необходимую для нахождения k кратчайших путей до целевой вершины. В работе [21] было показано, что алгоритм LVEA работает существенно быстрее не только алгоритма EA, но и во многих случаях быстрее алгоритма REA.

Общим недостатком алгоритмов EA, REA и LVEA является первый шаг, который требует построения всех кратчайших путей из начальной вершины (до всех остальных) с помощью алгоритма Дейкстры. Построение этих путей на графах, содержащих несколько десятков тысяч вершин (а именно такие графы обычно рассматриваются в практических задачах, связанных с построением траектории), чрезвычайно ресурсоемко. Для того чтобы снизить вычислительные затраты, был предложен алгоритм K* [10]. Идея алгоритма K* заключается в использовании эвристического алгоритма A* вместо алгоритма Дейкстры и построении кратчайшего пути (с помощью A*) лишь из начальной вершины в целевую. Во время этого процесса также формируется часть графа $P(G)$. После того как путь найден, с помощью алгоритма Дейкстры из графа $P(G)$ извлекается k кратчайших путей. Если в графе $P(G)$ меньше чем k путей, алгоритм расширяет граф $P(G)$ за счет обхода б*о*льшего числа вершин в графе G. Операция повторяется до тех пор, пока не будет построено k кратчайших путей. Алгоритм K* имеет ту же асимптотическую оценку сложности, что и алгоритмы EA и LVEA, но в большинстве практических задач его показатели вычислительной эффективности существенно выше. И чем

больше размер исходного графа G, тем выше разница в этих показателях в пользу алгоритма K*.

Все рассмотренные алгоритмы применимы на различных типах графов, в том числе и на МТ-графах. Однако эти алгоритмы не применимы в контексте поставленной нами задачи, так как в принципе не могут строить траектории с учетом ограничений на динамику движения БЛА. Это обуславливает необходимость разработки нового метода поиска совокупности путей, применимого на МТ-графах и учитывающего ограничения на максимальный угол отклонения. Такой метод будет описан ниже.

3.2. АЛГОРИТМ MULTILIAN

Предлагаемый алгоритм поиска совокупности путей, удовлетворяющих ограничениям на максимальный угол отклонения (и тем самым косвенно удовлетворяющим ограничениям на динамику движения беспилотного летательного аппарата), является модификацией предложенного ранее алгоритма LIAN, впервые описанного в работе [7]. Алгоритм LIAN основывается на тех же принципах, что и известный в литературе по искусственному интеллекту алгоритм A* [19], и в упрощенном виде принцип работы алгоритма можно описать следующим образом (подробнее см. в работах [7, 36]).

Алгоритм осуществляет сфокусированный (с помощью эвристики) перебор вершин графа и поддерживает в памяти два списка вершин: список *OPEN*, в котором располагаются нерассмотренные вершины графа – кандидаты на дальнейшую обработку, и список *CLOSE* – множество уже рассмотренных вершин. На каждой итерации алгоритма из списка *OPEN* выбирается вершина $a = argmin_{a \in OPEN}\{f(a)\}$, где $f(a) = g(a) + h(a)$, где $g(a)$ – длина кратчайшего пути из s в a, найденного к текущему моменту (эта величина может изменяться в ходе работы алгоритма), $h(a)$ – эвристическая оценка длины пути из a в g ($h(a) = d(a, g)$). Возможно также использование других эвристик – см. например [6]. Далее для выбранной вершины генерируются потомки, т.е. такие вершины a', для которых выполняются следующие условия:

1. Вершина a' проходима ($a' \in A^+$).

2. Расстояние от вершины-потомка до вершины-родителя приблизительно равно заранее заданной константе Δ, где Δ – входной параметр алгоритма ($d(a, a') \approx \Delta$).
3. Угол между двумя секциями, которые образуют вершина, ее родитель и родитель родителя, не превышает по модулю заданного значения α_m ($\alpha(e_1, e_2) < \alpha_m$, $e_1 = <bp(a), a>$, $e_2 = (a, a')$, $bp(a)$ – родитель вершины a).
4. Между a и a' нет препятствий, другими словами – секция $<a, a'>$ является проходимой (т.е. $los(a, a') = true$, проверяется с помощью алгоритма Брезенхема).

Сгенерированные таким образом вершины добавляются в список *OPEN*, и процесс повторяется. Если на каком-то шаге окажется, что целевая вершина находится на расстоянии меньшем либо равном Δ от текущей вершины, для целевой вершины удовлетворяется ограничение на угол и между целевой и текущей вершиной отсутствуют препятствия, то алгоритм завершает свою работу – искомый путь найден (и может быть восстановлен с помощью родительских указателей). Если в процессе работы алгоритм список *OPEN* исчерпан, LIAN возвращает *failure*, т.е. сообщение о том, что пути не существует.

В работах [1, 7, 36] показано, что алгоритм LIAN является корректным, полным и оптимальным (что следует из того, что LIAN использует ту же стратегию поиска в пространства состояний, что и A*), однако возможность отыскания пути на конкретной карте (при фиксированном ограничении на угол отклонения) зависит от входного параметра Δ. Заметим также, что на практике построение кратчайших путей нецелесообразно в силу чрезмерного расходования вычислительных ресурсов. Для отыскания субоптимальных решений используется техника взвешивания эвристики, т.е. использование функции $f(a)$ вида $g(a) + w \cdot h(a)$, где $w \geq 1$ – вес эвристики. Как показывают результаты экспериментов, при весе эвристики равном 2 удается существенным образом (более чем в 2–3 раза) сократить пространство поиска и повысить вычислительную эффективность алгоритма (в частности, скорость работы), при этом качество отыскиваемых решений (длины построенных путей) снижается незначительно (менее 2–3%).

Перед тем как перейти к описанию модификации алгоритма, предназначенной для поиска k различных путей, введем дополнительные определения.

Пусть существует некоторое множество промежуточных вершин графа $Q = \{q_{xy}\}$, чьи координаты x и y могут принимать значения отличные от $[0, \ldots, m-1]$, $[0, \ldots, n-1]$ соответственно. В частности, координаты могут быть отрицательными. При этом будем считать, что вершины из множества Q входят в область определения метрической функции d, т.е. известен способ определения расстояния между вершинами из множества Q и/или вершинами **Gr**. Будем считать, что в множество Q входит L элементов.

Рассмотрим теперь более подробно функцию выбора вершины из списка *OPEN*:

(2) $f(a) = g(a) + w \cdot h(a)$.

Как было сказано выше, $g(a)$ – это длина кратчайшего пути из s в a, найденного к текущей итерации алгоритма. Этот путь представим в виде последовательности секций, или, что то же самое, в виде последовательности клеток (см. раздел 1). Воспользуемся последним представлением и обозначим вершины, входящие в путь, как p_i: $\pi(s, a) = \{p_1, \ldots, p_i, \ldots, p_K\}$, $p_1 = s$, $p_K = a$. Заметим, что нижние индексы используются для нумерации вершин в последовательности, а не для обозначения индексных координат клеток, как ранее. Модифицируем теперь эвристическую функцию $h(a)(= h(p_K))$ следующим образом:

(3) $h'(p_K, Q) = d(p_K, g) + \theta \sum_{l=1}^{L} \min_{i=\overline{1,K}} d(q_l, p_i)$.

Первое слагаемое – расстояние от текущей вершины до целевой, т.е. $h(p_K)$. Второе слагаемое есть сумма минимальных расстояний от каждой из вершин множества Q до вершин входящих в путь, умноженная на коэффициент θ. Геометрическая интерпретация этого слагаемого представлена на рис. 2.

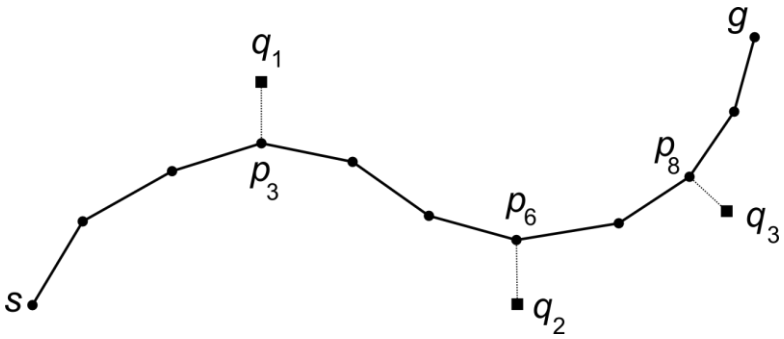

Рис. 2. Путь из s в g и промежуточные вершины (q_1, q_2, q_3)

На приведенном рисунке вершина p_3 является ближайшей вершиной к q_1, p_6 – к q_2, p_8 к q_3. Таким образом, для вершины p_8 имеем:

(4) $h'(p_8) = \theta[(q_1, p_3) + d(q_2, p_6) + d(q_3, p_8)]$.

Применение данной эвристической функции на практике приводит к такому порядку рассмотрения вершин графа, что, во-первых, минимизируется длина построенного пути (как ранее), во-вторых – построенный путь проходит как можно ближе к промежуточным вершинам. При этом следует заметить, что путь не обязан проходить непосредственно через них. Величина коэффициента θ определяет, насколько высоко будет цениться путь, проходящий близко (через) промежуточные вершины, по сравнению с просто кратчайшим путем. Если множество промежуточных вершин пусто, эвристика имеет в точности такой же вид, как и в оригинальном алгоритме.

Заметим, что использование определенной выше эвристической функции при поиске не нарушает свойств корректности и полноты, т.е. алгоритм завершает работу за конечное число шагов и находит путь, если он существует. Доказательство этих утверждений повторяет доказательство, приведенное в [1, 7, 36], поэтому ограничимся здесь лишь его кратким описанием. Доказательство первого утверждения основывается на том соображении, что на каждом шаге, независимо от эвристики, обрабатывается одна вершина графа, а так как число вершин ограничено, алгоритм завершит работу за конечное число ша-

гов. Доказательство второго утверждения основывается на том, что, если путь существует, то рано или поздно (за определенное число шагов) алгоритм рассмотрит все вершины, образующие его, а следовательно, найдет и сам путь. Следует, однако, заметить, что на практике при решении некоторых задач это число шагов может быть весьма велико.

Теперь приведем описание алгоритма MultiLIAN – алгоритма построения множества путей с использованием промежуточных вершин графа:

1. Построить путь из s в g с помощью алгоритма LIAN.
2. Определить множество промежуточных вершин.
3. Построить путь из s в g с учетом промежуточных вершин (т.е. используя эвристическую функцию описанную выше) – альтернативный путь.
4. Если построено заданное число путей, завершить алгоритм.
5. Вернуться к пункту 2.

Псевдокод алгоритма MultiLIAN приведен на рис. 3. Заметим, что во многом этот код идентичен LIAN (см. [28]). Основные различия: итерация по заданному (в качестве входного параметра) числу путей $\boldsymbol{k}$ (строка 3), наличие процедуры выбора промежуточных вершин (строка 4), способ приоритизации списка *OPEN* с учетом выбранных промежуточных вершин (строка 8–9), наличие переменной *PATHS* – списка, содержащего все построенные пути.

Вид построенных алгоритмом альтернативных путей целиком зависит от того, каким образом выбираются промежуточные вершины. Здесь и далее будем рассматривать задачу построения 2 альтернативных путей, при этом будем считать, что для построения каждого из них используется одна промежуточная вершина (т.е. всего используются две различные промежуточные вершины). Предлагается определять их координаты следующим образом. Строится отрезок, соединяющий начальную и целевую вершины. На данном отрезке отмечается точка, находящаяся на расстоянии a от начальной вершины. От этой точки откладывается два перпендикулярных отрезка длины b, концы которых и определяют координаты промежуточных вершин, см. рис. 4. Значения a и b являются входными параметрами алгоритма MultiLIAN.

```
1.  MULTILIAN(start, goal, Δ, α_m, k)
2.    PATHS := ∅
3.    for i from 1 to k
4.      Q_i=GetIntermediateAttractors();
5.      bp([start]) := ∅; g([start]) := 0;
6.      OPEN := ∅, CLOSED := ∅;
7.      OPEN.push([start]);
8.      while OPEN.size > 0
9.        [a] := argmin_{[a]∈OPEN} f([a]);
10.       // f([a]) = g([a]) + h' ([a], Q_i)
11.       OPEN.remove([a]);
12.       if a = goal
13.         path_i=getPathFromParentPoint
              ers([a]);
14.         PATHS.push(path_i);
15.         continue to next path;
16.       CLOSED.push([a]);
17.       Expand([a, Δ, α_m]);
18.   if PATHS = ∅
19.     return "paths not found"
20.   return PATHS;
21. end

22. Expand([a], Δ, α_m)
23.   SUCC =
      calculateCircleSuccessors([a], Δ);
24.   if dist(a, goal) < Δ
25.     SUCC.push([goal]);
26.   for each [a'] ∈ SUCC
27.     if a' is un-traversable
28.       continue;
29.     if | α(⟨bp(a), a⟩, ⟨a, a'⟩) | > α_m
30.       continue;
31.     if LineOfSight(a, a') = false
32.       continue;
33.     for each [a''] ∈ CLOSED
34.       if a'=a'' and bp(a')=bp(a'')
35.         continue;
36.     g(a') = g(bp(a')) + dist(a', bp(a'))
37.     OPEN.push([a']);
38.   end
```

Рис. 3. Псевдокод алгоритма MultiLIAN

Описанный способ генерации множества промежуточных вершин подходит для построения трех траекторий: основной и двух альтернативных. В случае если требуется построение большего числа альтернативных траекторий, необходимо определить способ генерации промежуточных вершин исходя из условий задачи и свойств карты. Например, можно откладывать с каждой стороны отрезка $[s, g]$ несколько вершин, находящихся на разных расстояниях от отрезка, соединяющего начальную и целевую вершины. В целом, способ генерации промежуточных вершин может являться предметом отдельных исследований.

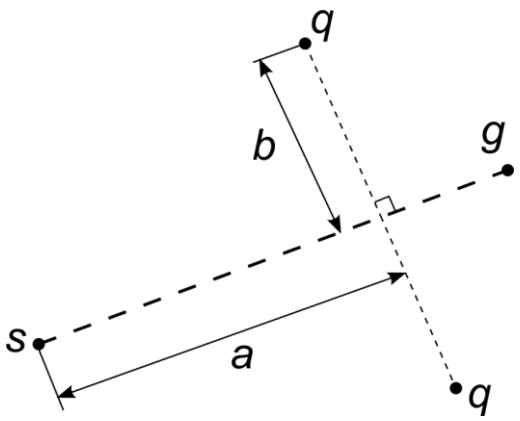

Рис. 4. Параметры a и b, определяющие расположение промежуточных вершин относительно начального и целевого расположения

Заметим, что в некоторых случаях MultiLIAN может строить альтернативные пути, идентичные первому построенному. Поясним на примере, когда множество промежуточных вершин Q для построения очередного альтернативного пути состоит из одной вершины q_1, и используется стратегия выбора этой вершины, описанная выше. Если препятствие большого размера находится на прямой, соединяющей начальную и целевую вершины, то найденный изначально путь может совпасть с путем, построенной с учетом промежуточных вершины, см. рис 5. Решить эту проблему можно различными способами выбора промежуточных вершин. В рамках данной работы такой подход не рассматривается, но может стать темой последующих исследований.

Представленный подход к построению множества путей может быть охарактеризован как метод грубой силы (англ. – brute force), так как на каждой итерации MultiLIAN альтернативный путь ищется без учета результатов предыдущих итераций. Очевидно, что последние могут быть использованы для повышения вычислительной эффективности алгоритма. Ниже будут описаны две модификации алгоритма MultiLIAN, которые

определенным образом учитывают результаты промежуточных итераций для сокращения пространства поиска.

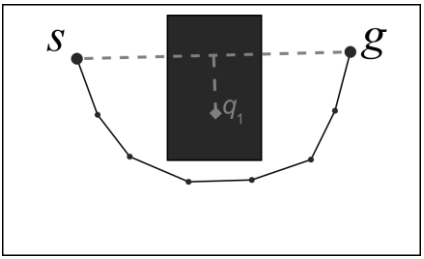

Рис. 5. Случай, когда альтернативный путь совпадает с первым построенным из-за расположения промежуточной вершины q_1

Итак, после завершения первой итерации алгоритма построен путь из *s* в *g*, а также образованы списки *OPEN* и *CLOSED*. Каждый элемент этих списков фактически определяет частичный путь до соответствующей вершины графа из начальной. При этом чем «сложнее» граф, на котором ищется путь, и чем больше на нем непроходимых клеток, тем больше вершин будет в этих списках (см. рис. 6).

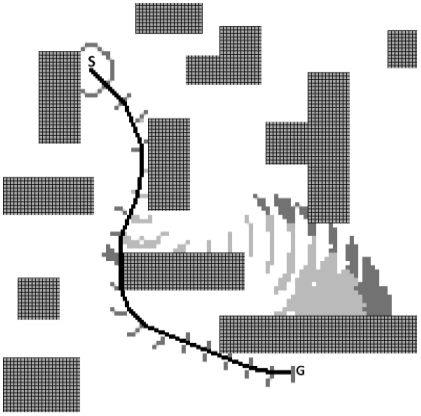

Рис. 6. Визуализация списков OPEN (*темно-серые вершины*) *и CLOSE* (*светло-серые вершины*) *после отыскания пути*

Логично в качестве промежуточных вершин выбирать вершины из множества $OPEN \cup CLOSE$, так как пути до этих вершин *уже* известны – в таком случае остается лишь достроить данный частичный путь (а не строить его полностью заново). Выбор вершин из множества $OPEN \cup CLOSE$ осуществляется, как и ранее, с помощью расчета расстояний до определенных промежуточных вершин. При этом при поиске альтернативного пути промежуточные вершины более не используются (т.е. используется оригинальная эвристика алгоритма LIAN).

Описанный подход получил название MultiLIAN-RU (RU – от англ. reuse, повторное использование). Наряду с очевидными достоинствами (повторное использование уже полученных на предшествующих итерациях данных) отметим и недостатки LIAN-RU. Во-первых, при использовании взвешенной эвристики (а на практике чаще всего используется именно она) число вершин в списках *OPEN* и *CLOSE* может быть невелико, что естественным образом ограничивает выбор. Во-вторых, путь из этих вершин до целевой может не существовать из-за угловых ограничений (начальная вершина нового поиска теперь характеризуется не только своими координатами, но и родителем, т.е. уже на первом шаге нового LIAN-поиска велика вероятность отсутствия вершин-потомков, удовлетворяющих ограничению на максимальный угол отклонения).

Другая модификация алгоритма MultiLIAN, использующая информацию предыдущих итераций, основывается на следующем предположении. После шага 1 алгоритма список *CLOSE* содержит в себе, наряду с вершинами, образующими искомый путь, вершины, на обработку которых потрачено время, но при этом не вошедшие в этот самый путь. Можно предположить, что последние являются «лишними» и при очередных итерациях поиска такие вершины должны быть исключены из рассмотрения (образно говоря таким вершинам, соответствуют «тупики» на карте местности, моделью которого служит рассматриваемый граф специальной структуры). Модифицированный соответствующим образом алгоритм будем называть MultiLIAN-CC (CC – от англ. exClusion from Close).

Псевдокод алгоритма MultiLIAN-CC и MultiLIAN-RU приведен на рис. 7 и 8. Заметим, что MultiLIAN-RU использует ту же процедуру Expand(), что и оригинальный MultiLIAN (поэтому на рисунке она не приведена), а MultiLIAN-CC – модифицированную процедуру.

```
1.  MULTILIAN-RU(start, goal, Δ, α_m, k)
2.      PATHS := ∅
3.      for i from 1 to k
4.          [start] = Get_start(i);
5.          OPEN := ∅, CLOSED := ∅;
6.          OPEN.push([start]);
7.          while OPEN.size > 0
8.              [a] := argmin_{[a]∈OPEN} f([a]);
9.              // f([a]) = g([a]) +h([a])
10.             OPEN.remove([a]);
11.             if a = goal
12.                 path_i=getPathFromParentPoint
                        ers([a]);
13.                 PATHS.push(path_i);
14.                 break;
15.             CLOSED.push([a]);
16.             Expand([a, Δ, α_m]);
17.         if i=1
18.             FormAlternativeStarts(CLOSE, k);
19.     if PATHS = ∅
20.         return "paths not found" ;
21.     return PATHS;
22. end

23. FormAlternativeStarts(CLOSE, k)
24.     for i from 2 to k
25.         q=GetFirstIntermediateAttractor();
26.         [start]_i := argmin_{[a]∈CLOSE} dist([a], q);
27. end

28. Get_start(i)
29.     if i = 1
30.         return [start];
31.     else
32.         return [start]_i;
33. end
```

Рис. 7. Псевдокод алгоритма MultiLIAN-RU

```
1. MULTILIAN-CC(start, goal, Δ, α_m, k)
2.     EXCLOSED := ∅; PATHS := ∅;
3.     for i from 1 to k
4.         bp([start]) := ∅; g([start]) := 0;
5.         OPEN := ∅, CLOSED := ∅;
6.         OPEN.push([start]);
7.         while OPEN.size > 0
8.             [a] := argmin_{[a]∈OPEN} f([a]);
9.             OPEN.remove([a]);
10.            if a = goal
11.                path_i=getPathFromParentPointer
                       s([a]);
34.                PATHS.push(path_i);
12.            CLOSED.push([a]);
13.            Expand([a, Δ, α_m]);
14.    if i=1
15.        EXCLOSED := CLOSED;
16. end

17. Expand([a], Δ, α_m)
18.    SUCC = calculateCircleSuccessors([a],
       Δ);
19.    if dist(a, goal) < Δ
20.        SUCC.push([goal]);
21.    for each [a'] ∈ SUCC
22.        if a' is un-traversable
23.            continue;
24.        if | α(⟨bp(a), a⟩, ⟨a, a'⟩) | > α_m
25.            continue;
26.        for each [a''] ∈ CLOSED
27.            if a'=a'' and bp(a')=bp(a'')
28.                continue;
29.        for each [a'''] ∈ EXCLOSED
30.            if a'=a'''
31.                continue;
32.        if LineOfSight(a, a') = false
33.            continue;
34.        OPEN.push([a']);
35.    end
```

Рис. 8. Псевдокод алгоритма MultiLIAN-CC

4. Экспериментальный анализ

4.1. МЕТОДИКА ПРОВЕДЕНИЯ ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

Для проведения экспериментальных исследований разработанного алгоритма MultiLIAN (а также его модификаций) рассматривалась модельная задача построения траектории маловысотного полета БЛА мультироторного типа AscTec Hummingbird [11] в городских условиях. Источником исходных данных выступала открытая геоинформационная система

OpenStreetMaps [30]. Графы для проведения экспериментов представляли собой модели фрагментов реальных городских карт (в частности, использовались карты города Москва). Размер отдельного фрагмента составлял 1347×1347 м^2, а соответствующий ГРД/МТ-граф имел размер 501×501, т.е. каждая вершина соответствовала области размером ~2,7×2,7 м^2. Выбор такого размера единичной области обусловлен физическими размерами мультикоптера AscTec Hummingbird, а также тем фактом, что обеспечить строгое следование по прямолинейной траектории не под силу ни одному известному управляющему регулятору, поэтому необходим некоторый пространственный запас (т.е. каждая единичная область больше по площади, чем площадь проекции объекта управления на рабочую плоскость). Подробнее о принципах выбора шага дискретизации карты сказано в [7]. Вершины, соответствующие зданиям, считались непроходимыми. Исследования проводились на двух коллекциях. В первой коллекции содержалось 80 различных графов, во второй – 100. На каждом графе было выбрано по 5 различных расположений начальной и целевой вершин, таким образом, общее число заданий составило 900. Сами же начальная и целевая вершины выбирались таким образом, что расстояние между ними было примерно одинаковым (образно говоря, начальное и целевое расположение беспилотного летательного аппарата выбирались на противоположных концах карт). Соотношение непроходимых вершин к общему числу вершин в графе (плотность препятствий) в обеих коллекциях составляет примерно 25%, однако в первой коллекции этот показатель варьируется в пределах от 15 до 35%, в то время как во второй коллекции этот показатель варьируется лишь в пределах 20–25% (т.е. вторая коллекция более однородна по плотности). На рис. 9 представлено изображение одного из использованных графов (линии сетки не показаны для наглядности, отмечены также найденные пути).

Рис. 9. Пример задания. Сплошной линией обозначен основной путь, пунктирной – альтернативные пути, найденные с использованием промежуточных вершин

Следуя рекомендациям работ [7, 36] (подтвержденным результатами предварительной серии экспериментов), алгоритм LIAN был инициирован следующим образом: вес эвристики равнялся двум ($w = 2$), параметр Δ равнялся пяти ($\Delta = 5$). Ограничение на максимальный угол отклонения составляло 25°. Такое ограничение (см. [7]) соответствует полету БЛА мультироторного типа AscTec Hummingbird [11] на скорости 4,5 м/с.

На выполнение отдельного задания было установлено временное ограничение – 60 секунд. Если алгоритм не укладывался в отведенное время, то результатом работы считалось *failure* (что учитывалось при подсчете значений интегральных показателей качества работы алгоритма – см. ниже).

Экспериментальное исследование проводилось на персональном компьютере архитектуры x86 с процессором iCore2Quad 2,5 ГГц и 2 Гб RAM, работающим под управлением

ОС Windows 7 64bit SP1. Программная реализация алгоритмов выполнена на языке C++.

В каждом эксперименте отслеживались значения следующих трех выходных параметров алгоритма:

1. M – (от англ. memory) число вершин, рассмотренных (и сохранённых в оперативной памяти) алгоритмом. Это число характеризует емкостную эффективность алгоритма.
2. T – (от англ. time) количество секунд, которое понадобилось алгоритму для нахождения путей. Показатель временной эффективности алгоритма.
3. PL – (от англ. path length) средняя длина найденных путей.

Также в каждой серии экспериментов отслеживались следующие интегральные показатели:

1. SR – (от англ. success rate) процент успеха, равный соотношению количества успешно выполненных заданий к их общему числу (900).
2. UPR – (от англ. unique path rate) процент уникальных путей, равный соотношению уникальных найденных путей к их общему (максимально возможному) числу (900 * 3 = 2700).
3. PAR-10 – (от англ. penalized average time) интегральная метрика, отражающая способность алгоритма быстро отыскивать решения задачи планирования в серии экспериментов [20]. PAR-10 = $(\Sigma T_1 + \Sigma T_2)/N$, где N – общее число экспериментов в серии (т.е. 900); T_1 – суммарное время по тем экспериментам, когда алгоритм завершил работу в установленный лимит времени (60 с); T_2 – число экспериментов, которые были прерваны (т.е. не уложились в 60 с), умноженное на 60 (временной лимит), умноженное на 10 (величина штрафа за «неспособность найти решение в отведенное время»).

4.2. *ПЕРВАЯ СЕРИЯ ЭКСПЕРИМЕНТОВ*

В первой серии экспериментов исследовалось влияние расположения промежуточных вершин (определяемого с помощью параметров a и b – см. раздел 2) на эффективность MultiLIAN. В таблице 1 приведены значения интегральных показателей, при этом цифры в названии алгоритма соответствуют значениям

параметров *a* и *b*, использованным при выборе промежуточных вершин.

Таблица 1. Интегральные показатели эффективности алгоритма MultiLIAN при различном расположении промежуточных вершин

	Moscow Maps 1.0			Moscow Maps 2.0		
	SR	UPR	PAR-10	SR	UPR	PAR-10
MultiLIAN 0,5-0,5	94,5%	94,5%	29,55	97,6%	97,6%	12,91
MultiLIAN 0,25-0,25	97%	96,67%	14,70	98%	97,53%	10,27
MultiLIAN 0,5-0,25	96%	96%	20,45	97,8%	97,8%	11,64
MultiLIAN 0,75-0,25	95,25%	95,17%	25,22	98%	97,93%	10,40

Полученные результаты демонстрируют явное преимущество использования параметров «0,25-0,25» и «0,5-0,25». MultiLIAN «0,25-0,25» показал лучший результат по критериям SR и PAR-10 на обеих коллекциях. Однако в некоторых случаях алгоритм с этими настройками находит несколько идентичных путей, в связи с чем показатель UPR ниже, чем SR. Подобная ситуация возникает и при использовании параметров «0,75-0,25». Заметим также, что на второй коллекции все 4 алгоритма продемонстрировали близкие результаты.

Проанализируем более детально критерий SR, а именно, рассмотрим график зависимости успешно выполненных заданий (по обеим коллекциям) от количества времени, которое потребовалось, чтобы завершить каждое задание (см. рис. 10).

График на рис. 10, как и предыдущие результаты, демонстрирует преимущество MultiLIAN с параметрами «0,25-0,25». 63,3% заданий алгоритм MultiLIAN «0,25-0,25» успешно завершил менее чем за 0,1 с. Стоит заметить, что аналогичный результат продемонстрировал MultiLIAN «0,5-0,25». Также интерес представляет количество успешно выполненных заданий,

завершившихся менее чем за 1 мин. У алгоритма MultiLIAN с параметрами «0,25-0,25» этот показатель равен 97,56%, MultiLIAN «0,5-0,25» – 97%, MultiLIAN «0,75-0,25» – 96,8%, MultiLIAN «0,5-0,5» – 96,2%.

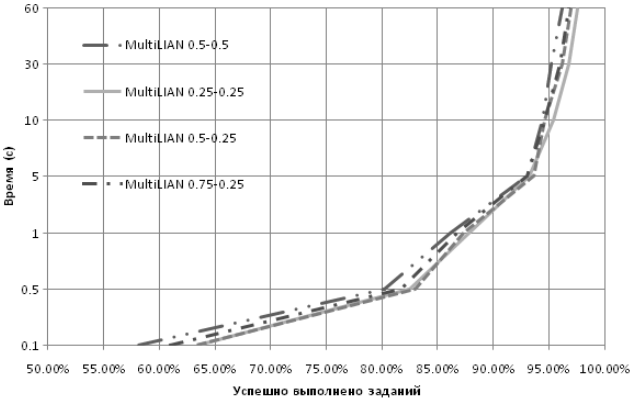

Рис. 10. График зависимости количества выполненных заданий от времени работы алгоритма

В таблице 2 представлены значения индикаторов *M*, *T* и *PL*. Данные получены путем усреднения результатов успешно выполненных заданий, т.е. для каждого алгоритма усреднялись только результаты успешно выполненных этим алгоритмом заданий.

Таблица 2. Индивидуальные показатели эффективности алгоритма MultiLIAN при различном расположении промежуточных вершин.

	Moscow Maps 1.0			Moscow Maps 2.0		
	M	*PL*	*T*	*M*	*PL*	*T*
MultiLIAN 0,5-0,5	12023	715,4	1,102	8727	593,8	0,927
MultiLIAN 0,25-0,25	11039	648,3	1,222	7350	542,0	0,684
MultiLIAN 0,5-0,25	10772	652,4	0,983	7718	545,2	0,853

	Moscow Maps 1.0			Moscow Maps 2.0		
	M	*PL*	*T*	*M*	*PL*	*T*
MultiLIAN 0,75-0,25	12363	664,8	1,277	8272	555,5	0,815

Результаты, приведенные в таблице 2, соответствуют сделанным ранее выводам о том, что почти по всем показателям на обеих коллекциях наиболее эффективно использование параметров «0,25-0,25». Также хорошие результаты демонстрирует алгоритм MultiLIAN с параметрами «0,5-0,25». Самые низкие показатели демонстрируются алгоритмом при использовании параметров «0,5-0,5». Объяснить это можно тем, что в этом случае промежуточные вершины находятся дальше всего от прямой между начальной и целевой вершинами, вследствие чего алгоритму необходимо рассмотреть больше вершин и потратить на это больше времени (по этой же причине средняя длина пути существенно больше по сравнению с аналогами).

Усредненные по обеим коллекциям и нормированные результаты представлены в виде гистограммы на рис. 11.

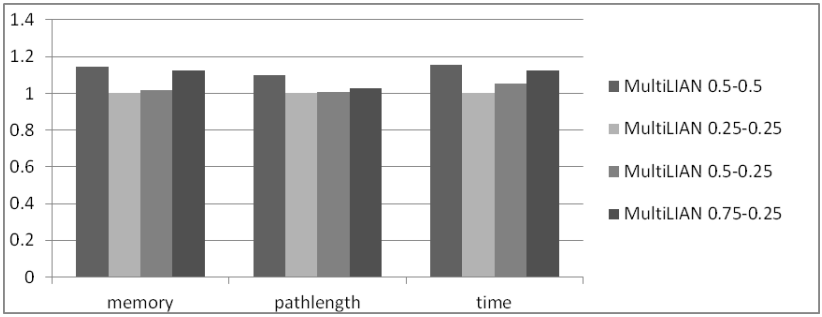

Рис. 11. Гистограммы нормированных усредненных значений индивидуальных индикаторов эффективности алгоритма MultiLIAN при различном расположении промежуточных вершин

Из рис. 6 видно, что алгоритм MultiLIAN с параметрами «0,5-0,5» тратит в среднем на 14% больше памяти и на 15% больше времени, чем MultiLIAN «0,25-0,25». Средняя длина

найденных путей у MultiLIAN «0,5-0,5» больше на 10%, а значения длины пути у остальных алгоритмов практически идентичны. Алгоритм MultiLIAN с параметрами «0,75-0,25» хуже, чем MultiLIAN «0,25-0,25» по показателям временной и ёмкостной эффективности на 12,5%.

Таким образом, в результате проведения первой серии экспериментов удалось определить наилучший способ выбора промежуточных вершин для построения альтернативных путей, а именно – наилучший способ инициализации входных параметров алгоритма MultiLIAN, отвечающих за расположение этих вершин: «0,5-0,25» и «0,25-0,25». При проведении дальнейших экспериментальных исследований использовались лишь эти варианты инициализации.

4.3. ВТОРАЯ СЕРИЯ ЭКСПЕРИМЕНТОВ

Во второй серии экспериментов сравнивались алгоритмы MultiLIAN и MultiLIAN-RU. Задания и настройки алгоритмов аналогичны тем, что были использованы ранее. Параметры выбора промежуточных вершин были установлены в соответствие с результатами первой серии экспериментов («0,5-0,25» и «0,25-0,25»). Ниже представлена таблица индивидуальных показателей эффективности.

Таблица 3. Индивидуальные показатели эффективности алгоритмов MultiLIAN и MultiLIAN-RU

	Moscow Maps 1.0			Moscow Maps 2.0		
	M	*PL*	*T*	*M*	*PL*	*T*
MultiLIAN 0,25-0,25	11039	648,3	1,215	7350	542,0	0,703
MLIAN-RU 0,25-0,25	6079	603,6	0,639	3992	503,6	0,404
MultiLIAN 0,5-0,25	10772	652,4	0,983	7718	545,2	0,853
MLIAN-RU 0,5-0,25	6194	604,5	0,608	4254	504,7	0,401

Основываясь на полученных данных, можно сделать вывод о том, что модификация MultiLIAN-RU существенно улучшает эффективность работы алгоритма. Затраты по памяти уменьшаются на ~80%, затраты по времени уменьшаются на ~60–110% в зависимости от коллекции и выбранных параметров. Средняя длина пути сокращается на ~8% (что весьма существенно). Однако при анализе интегральных результатов – см. таблицу 4 – становится ясно, что вывод сделан поспешно.

Таблица 4. Интегральные показатели эффективности алгоритмов MultiLIAN и MultiLIAN-RU

	Moscow Maps 1.0			Moscow Maps 2.0		
	SR	UPR	PAR-10	SR	UPR	PAR-10
MultiLIAN 0,25-0,25	97%	96,67%	14,68	98%	97,53%	10,29
MLIAN-RU 0,25-0,25	97,75%	37,25%	9,63	97,8%	35,07%	11,19
MultiLIAN 0,5-0,25	96%	96%	20,45	97,8%	97,8%	11,66
MLIAN-RU 0,5-0,25	97,5%	37,5%	11,09	98%	36,27%	9,99

Как видно из таблицы в большинстве случаев алгоритм MultiLIAN-RU не находит альтернативные траектории (см. показатель UPR). Другими словами, алгоритм (в подавляющем большинстве случаев) корректно завершается за отведенное время, однако все три найденных пути совпадают. При более детальном анализе была выяснена причина такого поведения алгоритма. Дело в том, что при использовании взвешенной эвристики размер списков *OPEN* и *CLOSE* (из которых алгоритмом MultiLIAN-RU выбираются вершины для построения альтернативных путей) крайне невелик и ближайшие вершины к промежуточным целям входят в *уже* построенный путь. Таким образом, алгоритм MultiLIAN-RU зачастую строит несколько абсолютно идентичных путей. Во многих случаях алгоритму в принципе не удается найти альтернативные пути, что связано с тем, что вершина, выбираемая из множества $OPEN \cup CLOSE$,

уже имеет родителя и из-за ограничения на угол, заданного родителем, алгоритм не может обойти ближайшие препятствия. В целом результаты экспериментальных исследований MultiLIAN-RU подтверждают справедливость высказанных ранее (см. раздел 2) опасений, по поводу возможной неэффективности его работы.

4.4. ТРЕТЬЯ СЕРИЯ ЭКСПЕРИМЕНТОВ

В третьей серии экспериментов сравнивались алгоритмы MultiLIAN и MultiLIAN-CC. Задания и настройки алгоритмов аналогичны тем, что были использованы ранее. Параметры выбора промежуточных вершин были установлены в соответствие с результатами первой серии экспериментов («0,5-0,25» и «0,25-0,25»). Ниже представлена таблица интегральных показателей эффективности.

Таблица 5. Интегральные показатели эффективности алгоритмов MultiLIAN и MultiLIAN-CC

	Moscow Maps 1.0			Moscow Maps 2.0		
	SR	UPR	PAR-10	SR	UPR	PAR-10
MultiLIAN 0,25-0,25	97%	96,67%	14,68	98%	97,53%	10,29
MLIAN-CC 0,25-0,25	97,75%	96,5%	10,12	98,2%	97,2%	8,93
MultiLIAN 0,5-0,25	96%	96%	20,45	97,8%	97,8%	11,66
MLIAN-CC 0,5-0,25	96,5%	96,5%	17,49	98%	98%	10,24

Как видно из представленных данных, MultiLIAN-CC (как и MultiLIAN) при значениях входных параметров «0,25-0,25» не всегда находит различные пути, в связи с чем значение показателя UPR несколько ниже значения SR. Впрочем, это не столь критично, как в случае с алгоритмом MultiLIAN-RU, так как при использовании параметров «0,5-0,25» уже все найденные альтернативные пути различны. В целом, если сравнивать MultiLIAN с MultiLIAN-CC при одинаковой параметризации, то

можно сделать вывод о том, что MultiLIAN-CC более эффективен, чем базовый алгоритм. Этот вывод подтверждается и при детальном анализе зависимости количества успешно выполненных заданий от времени работы (см. рис. 12).

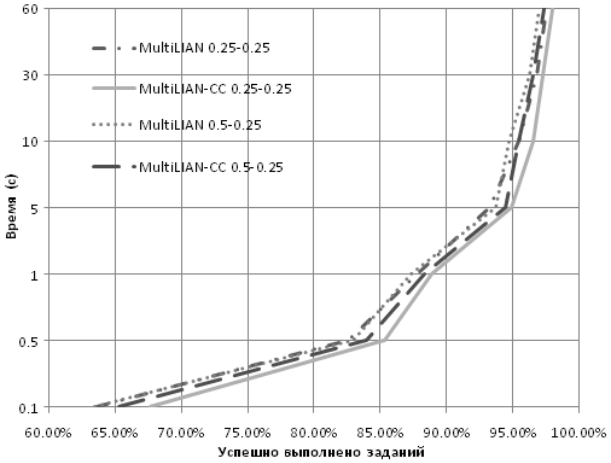

Рис. 12. График зависимости количества выполненных заданий от затраченного времени для алгоритмов MultiLIAN и MultiLIAN-CC

На временном интервале алгоритма MultiLIAN-CC демонстрирует своё преимущество. MultiLIAN-CC с параметрами «0,25-0,25» успешно завершает 67,7% заданий менее чем за 0,1 с; 95% всех заданий MultiLIAN-CC «0,25-0,25» обрабатывает менее чем за 5 с. Базовому алгоритму требуется как минимум 10 с, чтобы преодолеть порог в 95% успешно обработанных заданий.

В таблице 6 представлены значения индикаторов *M*, *T* и *PL*, полученных в результате проведения третьей серии экспериментов. Как и раньше, при подсчете значений индикаторов усреднялись только результаты успешно выполненных заданий.

Как видно из таблицы, значения средней длины пути у MultiLIAN и MultiLIAN-CC при одинаковых настройках практически идентичны, затраты по памяти также различаются не

сильно (в пределах 5,5% на первой коллекции и 3,4% – на второй), но при этом выигрыш MultiLIAN-CC по времени более существенен и составляет в среднем 15%. Более наглядно результаты эксперимента представлены на рис. 13 в виде гистограмм нормированных усредненных значений.

Таблица 6. Индивидуальные показатели эффективности алгоритмов MultiLIAN и MultiLIAN-CC

	Moscow Maps 1.0			Moscow Maps 2.0		
	M	*PL*	*T*	*M*	*PL*	*T*
MultiLIAN 0,25-0,25	11039	648,3	1,215	7350	542,0	0,703
MLIAN-CC 0,25-0,25	11528	648,4	1,136	7107	541,8	0,543
MultiLIAN 0,5-0,25	10772	652,4	0,983	7718	545,2	0,853
MLIAN-CC 0,5-0,25	11359	651,7	1,026	7797	544,52	0,653

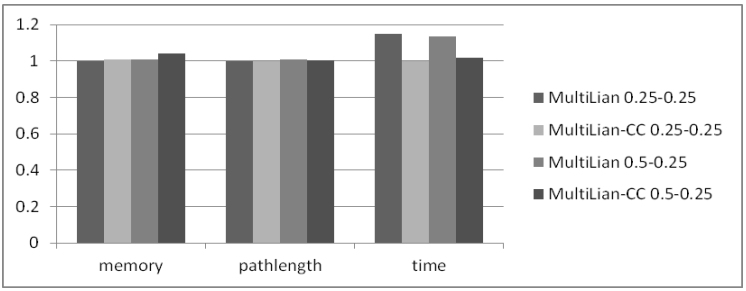

Рис. 13. Гистограммы нормированных усредненных значений индивидуальных индикаторов эффективности алгоритмов MultiLIAN и MultiLIAN-CC

В результате анализа результатов третьей серии экспериментов можно сделать вывод о том, что модификация MultiLIAN-CC превосходит базовый алгоритм по всем рассмотренным показателям эффективности.

4.5. ВЫВОДЫ

В результате проведенных экспериментальных исследований можно сделать следующие выводы:

1. Эффективность алгоритма MultiLIAN (по рассмотренным критериям) существенным образом зависит от выбора промежуточных вершин, которые используются для построения альтернативных путей. При этом на практике целесообразно использование следующих входных параметров MultiLIAN, отвечающих за расположение этих вершин: «0,5-0,25» и «0,25-0,25».

2. Использование модификации MultiLIAN-RU на практике нецелесообразно: при существенно более эффективном расходовании вычислительных ресурсов процент различных путей (т.е. путей отличающихся хотя бы одной вершиной), отыскиваемых алгоритмом, весьма низок.

3. Наиболее эффективным алгоритмом поиска различных альтернативных путей является MultiLIAN-CC, что свидетельствует о перспективности идеи итерационного использования результатов промежуточных вычислений для сокращения пространства поиска.

5. Заключение

В работе рассмотрена одна из ключевых подзадач, возникающих при планировании совокупности неконфликтных пространственных траекторий для коалиции беспилотных летательных аппаратов, а именно, задача автоматического планирования совокупности различных траекторий для отдельного аппарата. Предложен новый метод решения этой задачи, который, в отличие от известных аналогов, позволяет отыскивать траектории особого вида, косвенно учитывающие ограничения на динамику полета. Разработаны также модификации этого метода. Все описанные в работе методы программно реализованы, и проведено экспериментальное исследование алгоритмов в задаче автоматического планирования альтернативных траекторий маловысотного полета мультироторного беспилотного летательного аппарата в городских условиях. Результаты экспериментов показывают целесообразность применения предложен-

ного метода в таких задачах. К дальнейшим направлениям исследований можно отнести повышение вычислительной эффективности алгоритма при решении практических задач, а так же разработку на основе имеющегося задела метода планирования траекторий для коалиции летательных аппаратов, обладающих сложной динамикой движения.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 15-37-20893.

Литература

1. БАСКИН Е.С. *Алгоритм планирования пути lian-search, доказательство его сходимости, полноты и оптимальности* // Задачи современной информатики. – М.: ИПИ РАН, 2014. – С. 59–67.
2. ЖУЛЕВ В.И., ЛЕВУШКИН В.С., НГУЕН Т.Н. *Планирование локальной траектории автомобиля-робота в реальном времени* // Вестник РГРТУ. – 2013. – №4(46). Часть 3. – С. 18–23.
3. МАКАРОВ Д.А., ПАНОВ А.И., ЯКОВЛЕВ К.С. *Архитектура многоуровневой интеллектуальной системы управления беспилотными летательными аппаратами* // Искусственный интеллект и принятие решений. – 2015. – №3. – С. 18–33.
4. ОСИПОВ Г.С., ТИХОМИРОВ И.А., ХАЧУМОВ В.М. и др. *Интеллектуальное управление транспортными средствами: стандарты, проекты, реализации* // Авиакосмическое приборостроение. – 2009. – №6. – С. 34–43.
5. ЯКОВЛЕВ К.С., БАСКИН Е.С. *Графовые модели в задаче планирования траектории на плоскости* // Искусственный интеллект и принятие решений. – 2013. – №1. – С. 5–12.
6. ЯКОВЛЕВ К.С., БАСКИН Е.С. *Эвристическая оценка качества построения пути методом LIAN-Search* // Шестая Международная конференция «Системный анализ и информационные технологии» САИТ-2015 (15-20 июня

2015 г., г. Светлогорск, Россия): Труды конференции. В 2-х т. – Т. 1. – С. 132–138.
7. ЯКОВЛЕВ К.С., МАКАРОВ Д.А., БАСКИН Е.С. *Метод автоматического планирования траектории беспилотного летательного аппарата в условиях ограничений на динамику полета* // Искусственный интеллект и принятие решений. – 2014. – №4. – С. 3–17.
8. ЯКОВЛЕВ К.С., МАКАРОВ Д.А., ПАНОВ А.И. и др. *Принципы построения многоуровневых архитектур систем управления беспилотными летательными аппаратами* // Авиакосмическое приборостроение. – 2013. – №4. – С. 10–28.
9. ALBUS J. et al. *4D/Real-time Control System* (4D/RCS): *A Reference Model Architecture for Unmanned Vehicle Systems* v.2.0, NIST, NISTIR 6910, 2002.
10. ALJAZZAR H., LEUE S. *K*: A heuristic search algorithm for finding the k shortest paths* // Journal of Artifcial Intelligence. – 2011. – Vol. 175. – P. 2129–2154.
11. *AscTec Hummingbird* – [Электронный ресурс]. – URL: http://www.asctec.de/en/uav-uas-drone-products/ asctec-hummingbird/ (дата обращения: 13.11.2015).
12. BERTRAM D. et al. *An integrated approach to inverse kinematics and path planning for redundant manipulators* // Proc. IEEE International Conference on Robotics and Automation (ICRA 2006). – 2006. – P. 1874–1879.
13. BRESENHAM J.E. *Algorithm for computer control of a digital plotter* // IBM Systems Journal. – 1965. – No. 4(1). – P. 25–30.
14. BRISTEAU P.-J., CALLOU F., VISSIERE D. et al. *The Navigation and Control Technology Inside the AR.Drone Micro UAV* // Preprints of the 18th IFAC World Congress (Milano, Italy). Milano: IFAC, 2011. – Vol. 18, Part 1. – P. 1477–1484.
15. DIJKSTRA E.W. *A note on two problems in connexion with graphs* // Numerische mathematik. – 1959. – Vol. 1. №1. – P. 269–271.
16. EPPSTEIN D. *Finding the k shortest paths* // SIAM Journal on Computing. – 1998. – No. 28(2). – P. 652–673.

17. FREDERICKSON G.N. *An optimal algorithm for selection in a min-heap* // Information and Computation. – 1993. – No. 104(2). – P. 197–214.
18. HARABOR D., GRASTIEN A.. *Online graph pruning for pathfinding on grid maps*. 2011 In AAAI-11 – [Электронный ресурс]/ – URL: http:// grastien.net /ban/articles /hg-aaai11.pdf (дата обращения: 13.11.2015).
19. HART P.E., NILSSON N.J., RAPHAEL B. *A formal basis for the heuristic determination of minimum cost paths* // IEEE Trans. on Systems Science and Cybernetics. – 1968. – No. 4(2). – P. 100–107.
20. HUTTER F., HOOS H.H., LEYTON-BROWN K. et al. *ParamILS: an automatic algorithm configuration framework* // Journal of Artificial Intelligence Research. – 2009. – No. 36(1). – P. 267–306.
21. JIM´ENEZ V.M., MARZAL A. *A lazy version of Eppstein's K shortest paths algorithm* // Experimental and Efficient Algorithms. – 2003. – Vol. 2647. – P. 179–191.
22. JIM´ENEZ V.M., MARZAL A. *Computing the k shortest paths: A new algorithm and an experimental comparison* // Algorithm Engineering – 1999. – Vol. 1668. – P. 15–29.
23. KIM H., KIM D., SHIN J.U. et al. *Angular rate-constrained path planning algorithm for unmanned surface vehicles* // Ocean Engineering. – 2014. – No. 84. – P. 37–44.
24. KIM H., PARK B., MYUNG H. *Curvature Path Planning with High Resolution Graph for Unmanned Surface Vehicle* // Robot Intelligence Technology and Applications. – 2012. – Vol. 208. – P. 147–154.
25. LAVALLE S.M., HUTCHINSON S.A. *Optimal motion planning for multiple robots having independent goals* // IEEE Transactions on Robotics and Automation. – 1998. – Vol. 14, Issue 6. – P. 912–925.
26. LAVALLE S.M., KUFFNER JR J.J. *Rapidly-exploring random trees: Progress and prospects* // Proc. Workshop on the Algorithmic and Computational Robotics: New Directions. – 2000. – [Электронный ресурс] – URL: http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf (дата обращения: 13.11.2015).

27. LIKHACHEV M., STENTZ A. *R* Search* // Proc. Twenty-Third AAAI Conference on Artificial Intelligence. Menlo Park, Calif.: AAAI press. – 2008. – [Электронный ресурс] – URL: http://repository.upenn.edu/cgi/viewcontent.cgi?article=1030&context=grasp_papers (дата обращения: 13.11.2015).
28. *Mikrokopter* – [Электронный ресурс] – URL: http://mikrokopter.de (дата обращения 13.11.2015).
29. NASH A., DANIEL K., KOENIG S. *Theta*: Any-Angle Path Planning on Grids* // Proc. National Conference on Artificial Intelligence – 2007. – Vol. 22, No. 2. – P. 1177–1183.
30. *OpenStreetMaps* – [Электронный ресурс] – URL: http://wiki.openstreetmap.org/wiki/Database (дата обращения: 13.11.2015).
31. *Parrot Ar.Drone* – [Электронный ресурс] – URL: http://ardrone2.parrot.com/ (дата обращения: 13.11.2015).
32. SILVER D. *Cooperative pathfinding* // AI Programming Wisdom. – 2006. – P. 99–111.
33. STANDLEY T. *Finding optimal solutions to cooperative pathfinding problems* // In AAAI. – 2010. – P. 173–178.
34. WANG K.-H.C., BOTEA A. *Fast and Memory-Efficient Multi–Agent Pathfinding* // ICAPS. – 2008. – P. 380–387.
35. WANG K.-H.C., BOTEA A. *Tractable Multi-Agent Path Planning on Grid Maps* // Proc. International Joint Conference on Artificial Intelligence (IJCAI). – 2009. – P. 1870–1875.
36. YAKOVLEV K., BASKIN E., HRAMOIN I. *Grid-based angle-constrained path planning* // Proc. 38th Annual German Conference on AI, Dresden, Germany, September 21-25, 2015. – Springer International Publishing. doi 10.1007/978-3-319-24489-1_16. – P. 208–221.

DYNAMICS CONSTRAINT-AWARE PLANNING OF MULTIPLE PATHS FOR UNMANNED VEHICLE

Konstantin Yakovlev, Institute for Systems Analysis of Federal Research Centre "Computer Science and Control" of Russian Academy of Sciences (ISA RAS), Cand.Sc. (yakovlev@isa.ru).
Egor Baskin, Institute for Systems Analysis of Federal Research Centre "Computer Science and Control" of Russian Academy of Sciences (ISA RAS), PhD student (baskin@isa.ru).
Anton Andreychuk, Peoples Friendship University of Russia, Moscow, student (andreychuk@mail.com).

Abstract: We study alternative paths' planning problem for an unmanned vehicle. It constitutes a key part of a bigger problem – that of multi-agent path planning, i.e., finding a non-conflicting path set for a coalition of vehicles. We propose a new path-planning technique, which indirectly takes into account vehicle movement dynamics and guarantees feasibly of resulting paths. We also elaborate a number of extensions of the method proposed. We conduct an empirical study of all introduced algorithms by running the large number of experiments simulating nap-of-the-earth flight of a compact multirotor unmanned aerial vehicle in urban environment.[1]

Keywords: unmanned vehicle, multirotor, quadrotor, navigation, intelligent control system, planning, path planning, path finding, alternative paths, k shortest paths, angle-constrained path planning, heuristic search.

Статья представлена к публикации
членом редакционной коллегии Н.И. Базенковым

Поступила в редакцию 28.10.2015.
Опубликована 30.11.2015.

---

[1] *The reported study was partially supported by RFBR, research project No. 15-37-20893.*